

CHAPTER 1

INTRODUCTION

This chapter gives an overview about the aim , objectives , features, background and operation environment of the system.

1.1 PROJECT AIMS AND OBJECTIVES

The project aims and objectives that will be achieved after completion of this project are discussed in this subchapter. The main aim and objectives are as follows:

AIM:

- To provide an easy and convenient way for customers to book movie tickets online
- To allow customers to browse movie schedules and select seats from the comfort of their own homes
- To reduce long waiting times and queues at cinema ticket counters
- To enable cinema staff to manage ticket sales and seat allocations more efficiently
- To increase revenue by providing an additional sales channel for movie tickets.

Objectives:

- To create a user-friendly interface that enables customers to search for movies, view schedules, and select seats easily
- To ensure that the system is secure and reliable, with appropriate measures to protect customer data and prevent fraud
- To integrate with existing cinema management systems to ensure that seat availability is updated in real time
- To enable customers to purchase tickets using a variety of payment methods, including credit/debit cards, mobile wallets, and online banking

1.2 FEATURES

- User friendly Interface
- User can Sign-up & Login into the website
- Movie selection and browsing by location or movie name
- Seat selection and reservation
- Booking the Tickets
- Then, User can Download E-Ticket

1.3 BACKGROUND OF PROJECT

- ✓ Industry context: Start by providing some context about the movie ticket booking industry. This might include information about the size of the market, recent trends, and key players in the industry.
- ✓ Problem statement: Identify the problem that the movie ticket booking system aims to solve. This might include issues such as long queues, inconvenient ticket booking processes, and limited availability of movie tickets.
- ✓ Goals and objectives: Explain the specific goals and objectives of the project, such as improving customer experience, increasing revenue, and reducing costs.
- ✓ Scope: Describe the scope of the project, including the specific features and functions that the movie ticket booking system will include. This might include things like seat selection, movie schedules, payment options, and customer support.
- ✓ Target audience: Identify the target audience for the movie ticket booking system, such as moviegoers, cinema staff, and management.
- ✓ Technology stack: Provide information about the technology stack that will be used to build the movie ticket booking system, including programming languages, frameworks, and third-party tools.

- ✓ **Timeline:** Outline the expected timeline for the project, including key milestones and deadlines.
- ✓ **Team structure:** Describe the structure of the project team, including roles and responsibilities, and any third-party vendors or contractors that will be involved.
- ✓ **Budget:** Provide a high-level overview of the project budget, including estimated costs for development, testing, and deployment.
- ✓ **Success metrics:** Finally, identify the success metrics that will be used to measure the effectiveness of the movie ticket booking system. This might include metrics such as customer satisfaction, ticket sales, and revenue growth.

1.4 OPERATION ENVIRONMENT

PROCESSOR	INTEL CORE PROCESSOR OR BETTER PERFORMANCE
OPERATING SYSTEM	WINDOWS 7 or Higher, UBUNTU
MEMORY	1GB RAM OR MORE
HARD DISK SPACE	MINIMUM 3 GB FOR DATABASE USAGE FOR FUTURE
DATABASE	SQLite3

CHAPTER 2

SYSTEM ANALYSIS

In this chapter, we will discuss and analyze about the developing process of movie ticket booking System including software requirement specification (SRS) and comparison between existing and proposed system. The functional and non-functional requirements are included in SRS part to provide complete description and overview of system requirement before the developing process is carried out.

2.1 SOFTWARE REQUIREMENT SPECIFICATION

2.1.1 GENERAL DESCRIPTION

PRODUCT DESCRIPTION:

Our movie ticket booking system is an easy-to-use online platform that enables customers to browse movie schedules, select seats, and purchase tickets from the comfort of their own homes. With our system, customers can view up-to-date information on show times and seat availability, and can choose their preferred seats from an interactive seating chart. The system also enables cinema staff to manage ticket sales and seat allocations more efficiently. With our movie ticket booking system, movie goers can enjoy a hassle-free booking experience and cinemas can increase revenue through an additional sales channel.

PROBLEM STATEMENT:

❖ Inefficient ticket purchasing process:

The traditional process of purchasing movie tickets can be slow and inconvenient for customers, especially during peak hours or popular movie releases. This can result in long queues and frustrated customers.

❖ Limited availability of seats:

The availability of seats for popular movies is often limited, which means that customers may miss out on their preferred screening times or seating options.

❖ Inaccurate seat allocation:

When seat allocations are done manually, there is a risk of errors in allocation, which can lead to double bookings or customer dissatisfaction.

❖ Inefficient ticket management:

Manual ticket management can lead to inefficient use of staff time and resources, as well as potential errors in reporting and record-keeping.

❖ Lack of customer insights:

Without a system to capture and analyze customer data, cinemas may miss out on opportunities to better understand their customers' preferences and behaviors, which can impact their overall business strategies.

2.1.2 SYSTEM OBJECTIVES

➤ User registration and authentication:

The system should allow users to register and create their profiles, and authenticate users when they log in.

➤ Movie listings and schedules:

The system should display all the movies playing in different theatres, along with their show timings and other relevant details.

➤ Seat selection and booking:

The system should allow users to select their preferred seats and book them for a particular show time.

➤ Payment processing:

The system should have a secure payment gateway to enable users to make online payments for their tickets.

➤ Confirmation and ticket generation:

Once the booking is confirmed and payment is processed, the system should generate a confirmation email or message and a printable ticket for the user.

2.1.3 SYSTEM REQUIREMENTS

2.1.3.1 NON FUNCTIONAL REQUIREMENTS

❖ **Performance:**

The system should be able to handle a large number of users and transactions simultaneously without any significant slowdowns or crashes. The response time for user interactions should be fast and efficient.

❖ **Reliability:**

The system should be highly reliable and available at all times. It should have backup and disaster recovery mechanisms in place to minimize downtime.

❖ **Security:**

The system should be secure to protect sensitive user information, such as credit card details, from unauthorized access, theft, or loss. It should comply with security standards and regulations.

❖ **Scalability:**

The system should be scalable to meet growing demands and changing user requirements. It should be able to handle increasing traffic, data volumes, and processing power.

❖ **Usability:**

The system should have a simple and intuitive user interface that is easy to navigate and use. It should also be accessible to people with disabilities.

❖ **Compatibility:**

The system should be compatible with various devices and browsers, ensuring that users can access it from any device or platform.

❖ **Maintainability:**

The system should be easy to maintain, update, and modify. The code should be well-documented, structured, and follow industry-standard coding practices.

❖ **Interoperability:**

The system should be able to integrate with other third-party systems or applications, such as payment gateways or customer relationship management (CRM) systems.

2.1.3.2 FUNCTIONAL REQUIREMENTS

❖ **User Registration:**

The system should allow users to create an account, fill in their personal details, and save their preferences.

❖ **Movie Listing:**

The system should display the list of movies, along with their descriptions, ratings, reviews, and trailers.

❖ **Show times and Theatres:**

The system should show the list of show times, theatres, locations, and the availability of seats for each movie.

❖ **Seat Selection:**

The system should allow users to select their preferred seats, the number of seats, and the type of seats.

❖ **Booking and Payment:**

The system should enable users to book their seats, confirm their booking, and make online payments for their tickets.

❖ **Ticket Generation:**

The system should generate electronic tickets, which the user can either print or show on their mobile devices.

2.2 EXISTING VS PROPOSED SYSTEM

Existing movie ticket booking systems usually offer basic functionality, such as movie listing, show times, seat selection, and online booking. However, they may lack features such as user accounts, loyalty programs, feedback and review systems, and advanced analytics.

In contrast, a proposed movie ticket booking system could offer additional functionality and enhancements to improve the customer experience and business operations. Some proposed features could include.

Personalized User Experience:

A proposed system could provide a more personalized user experience by displaying movie recommendations based on the user's viewing history and preferences.

Virtual Seat Selection:

A proposed system could enable users to see a virtual view of the theatre seating layout and select their preferred seats, providing a more immersive and interactive booking experience.

Social Media Integration:

A proposed system could integrate with social media platforms, allowing users to share their booking details with friends and family and promote the movies they have watched.

Mobile App:

A proposed system could offer a dedicated mobile app, enabling users to book tickets, receive notifications, and access loyalty programs and rewards on-the-go.

Advanced Analytics:

A proposed system could provide advanced analytics, such as real-time data on ticket sales, occupancy rates, and user demographics, helping theatre owners make data-driven decisions to improve their operations.

CHAPTER 3

SYSTEM SPECIFICATIONS

SOFTWARE AND HARDWARE REQUIREMENTS

This section describes the software and hardware requirements of the system

3.1 SOFTWARE REQUIREMENTS

Languages used:

Front End: HTML, CSS

Back End: Python (Flask)

Database: SQLite3

- Operating system - Windows 10 is used as the operating system as it is stable and supports more features and is more user friendly.
- Database SQLite3 is used as database as it is easy to maintain and retrieve Records by simple queries which are in English language which are easy to Understand and easy to write.
- Python is used to backend tool and flask is used to framework.
- Development tools and Programming language- HTML is used to write the whole code and develop Web pages with CSS for styling work

3.2 HARDWARE REQUIREMENTS

Processor: Intel core i5 2nd generation

Ram: 1GB

- Intel core i5 2nd generation is used as a processor because it is fast than other Processors and provide reliable and stable and we can run our pc for long time. By using this processor we can keep on developing our project without any worries.
- Ram 1 GB is used as it will provide fast reading and writing capabilities and Will in turn support in processing.

CHAPTER 4

SOFTWARE DESCRIPTION

The whole Project is divided in two parts the front end and the back end.

4.1 FRONT END

The front end is designed using of HTML , CSS.

➤ **HTML – HYPERTEXT MARKUP LANGUAGE**

HTML or Hyper Text Mark up Language is the main mark up language for creating web pages and other information that can be displayed in a web browser. HTML is written in the form of HTML elements consisting of tags enclosed in angle brackets (like <html>), within the web page content. HTML tags most commonly come in pairs like <h1> and </h1>, although some tags represent empty elements and so are unpaired, for example . The first tag in a pair is the start tag, and the second tag is the end tag (they are also called opening tags and closing tags). In between these tags web designers can add text, further tags, comments and other types of text-based content. The purpose of a web browser is to read HTML documents and compose them into visible or audible web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page. HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts written in languages such as JavaScript which affect the behaviour of HTML web pages.

➤ CSS – CASCADING STYLE SHEETS

Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a mark up language. While most often used to style web pages and interfaces written in HTML and XHTML, the language can be applied to any kind of XML document, including plain XML, SVG and XUL. CSS is a corner stone specification of the web and almost all web pages use CSS style sheets to describe their presentation. CSS is designed primarily to enable the separation of document content from document presentation, including elements such as the layout , colors , and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple pages to share formatting , and reduce complexity and repetition in the structural content (such as by allowing for table less web design). CSS can also allow the same mark up page to be presented in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser or screen reader) and on Braille-based, tactile devices. It can also be used to allow the web page to display differently depending on the screen size or device on which it is being viewed. While the author of a document typically links that document to a CSS file, readers can use a different style sheet, perhaps one on their own computer, to override the one the author has specified. However if the author or the reader did not link the document to a specific style sheet the default style of the browser will be applied. CSS specifies a priority scheme to determine which style rules apply if more than one rule matches against a particular element. In this so-called cascade, priorities or weights are calculated and assigned to rules, so that the results are predictable.

4.2 BACK END

The back end language used python designed using flask for framework and SQLite3 is used to design the Databases

PYTHON

Python is a high-level, interpreted programming language known for its simplicity, ease of use, and versatility. It was created in the late 1980s by Guido van Rossum and has since become one of the most popular programming languages in the world.

Python is designed to emphasize code readability and productivity, making it a popular choice for beginners and experienced programmers alike. Its syntax is concise and intuitive, with a focus on using whitespace indentation to indicate code blocks. This makes it easier to read, write, and maintain Python code.

Python has a vast standard library, which provides pre-written modules and functions for a wide range of programming tasks, including web development, data analysis, machine learning, scientific computing, and more. Python's popularity and flexibility have also led to the development of a large and active community of developers, who have created numerous third-party libraries and frameworks that extend the language's functionality.

Python is an interpreted language, which means that code is executed directly by an interpreter, without the need for a separate compilation step. This makes it easier to write and test code, as changes can be quickly made and tested without having to recompile the entire program.

Overall, Python's simplicity, versatility, and ease of use make it a popular choice for a wide range of applications, from web development to data science to automation and beyond.

Python is a high-level, interpreted programming language known for its simplicity, ease of use, and versatility. It was created in the late 1980s by Guido van Rossum and has since become one of the most popular programming languages in the world.

Python is designed to emphasize code readability and productivity, making it a popular choice for beginners and experienced programmers alike. Its syntax is concise and intuitive, with a focus on using whitespace indentation to indicate code blocks. This makes it easier to read, write, and maintain Python code.

Python has a vast standard library, which provides pre-written modules and functions for a wide range of programming tasks, including web development, data analysis, machine learning, scientific computing, and more. Python's popularity and flexibility have also led to the development of a large and active community of developers, who have created numerous third-party libraries and frameworks that extend the language's functionality.

Python is an interpreted language, which means that code is executed directly by an interpreter, without the need for a separate compilation step. This makes it easier to write and test code, as changes can be quickly made and tested without having to recompile the entire program.

Overall, Python's simplicity, versatility, and ease of use make it a popular choice for a wide range of applications, from web development to data science to automation and beyond.

Flask (frame work):

Flask is a micro web framework written in Python that allows developers to easily build web applications. It is called "micro" because it does not require particular tools or libraries and has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. It is simple, flexible, and lightweight, making it an ideal choice for small to medium-sized applications. Flask is built on top of the Werkzeug WSGI toolkit and the Jinja2 templating engine. Werkzeug is a utility library that provides useful tools for developing WSGI applications, while Jinja2 is a powerful template engine that allows developers to easily create HTML, XML, or other markup formats . Flask is also highly extensible, and developers can add extensions to enhance its functionality, such as Flask-RESTful for building RESTful APIs, Flask-SQLAlchemy for database integration, Flask-WTF for form validation, and many others. Overall, Flask is a popular choice among Python developers who want to build lightweight and flexible web applications quickly and easily.

SQLite3:

SQLite3 is a lightweight and open-source relational database management system (RDBMS) that uses a small C library and stores data in a self-contained, serverless, zero-configuration, and transactional database file. It is widely used in various applications and embedded systems that require a local or portable database solution, such as desktop software, mobile apps, web browsers, operating systems, IoT devices, and more. SQLite3 supports SQL syntax, ACID transactions, multiple tables, indexes, views, triggers, and other common database features. It is also compatible with various programming languages and platforms, including Python, Java, C++, PHP, .NET, and others

CHAPTER 5

SYSTEM DESIGN

5.1 TABLE DESIGN

VARIOUS TABLES TO MAINTAIN INFORMATION

➤ Table structure for table 'user'

Field	Data type	Default	Key	Extra
Id	integer	NOT NULL	Primary key	Auto increment
Username	Text	NOT NULL		
Password	Text	NOT NULL		

➤ Table structure for table 'Movies'

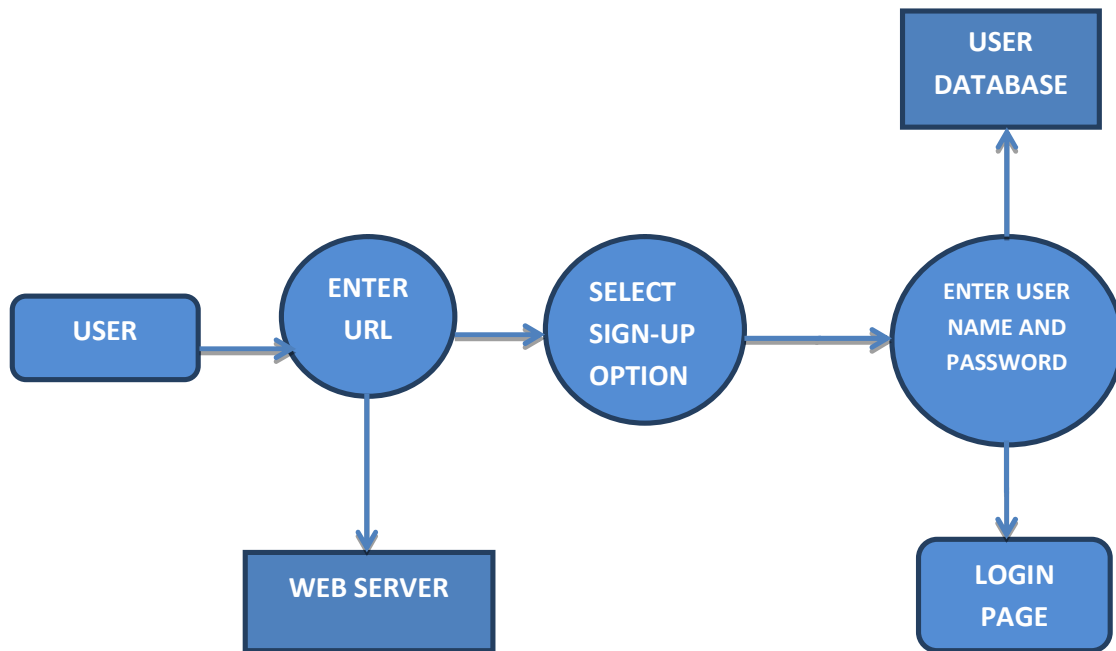
Field	Data type	Default	Key	Extra
movie_name	text	NOT NULL		
theatre_name	text	NOT NULL		
Location	text	NOT NULL		
Screen	text	NOT NULL		
showtime	text	NOT NULL		
available_seats	text	NOT NULL		
Id	integer	NOT NULL	Primary Key	Auto Increment

➤ Table structure for table 'booking'

Field	Data type	Default	Key	Extra
Id	integer	NOT NULL	Primary key	Auto increment
user_name	Text	NOT NULL		
movie_name	Text	NOT NULL		
theatre_name	Text	NOT NULL		
location_name	Text	NOT NULL		
showtime	Text	NOT NULL		
Screen	Text	NOT NULL		
seat_number	Text	NOT NULL		

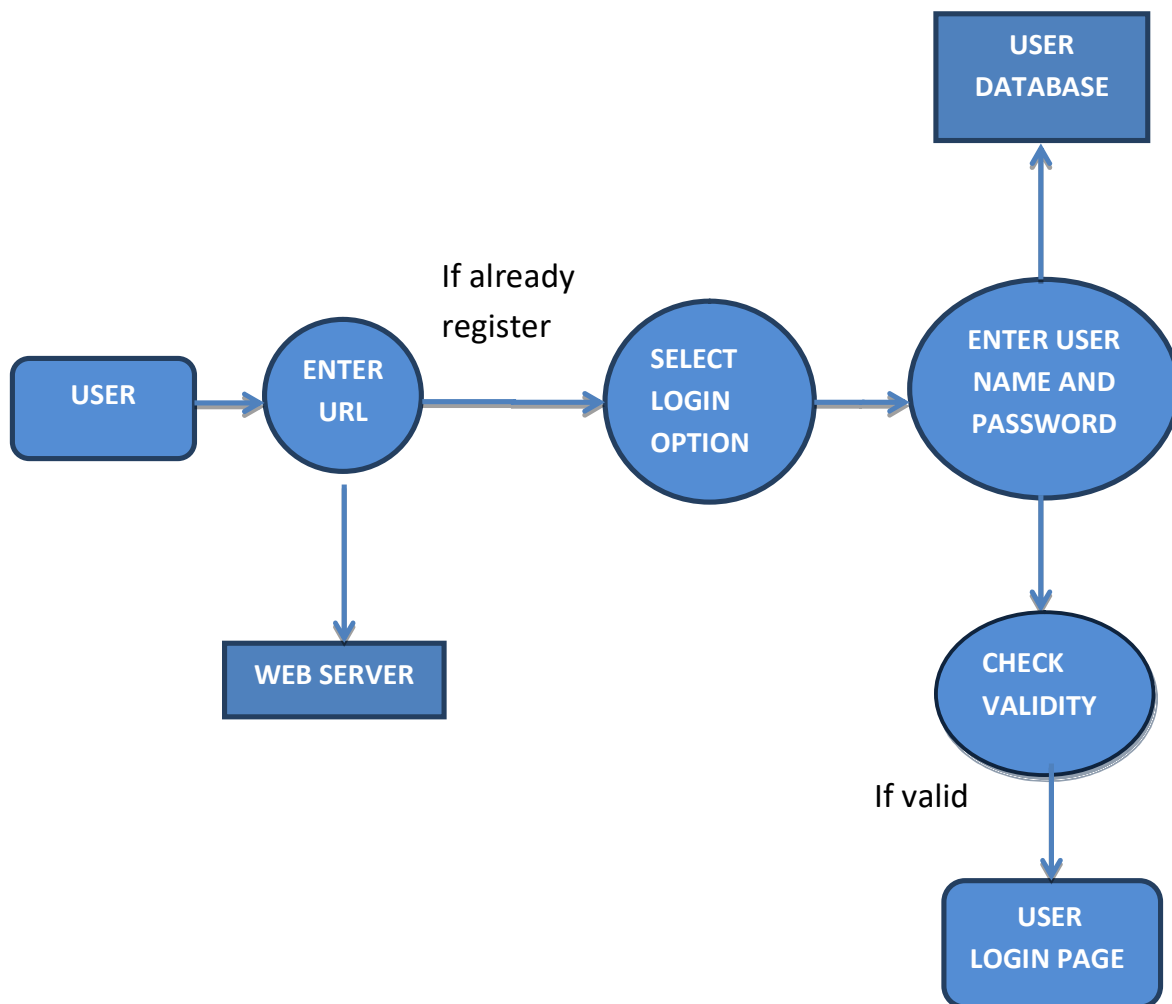
5.2 DATA FLOW DIAGRAM

5.2.1 DATA FLOW DIAGRAM FOR ADD NEW USER



After entering to the home page of the website, admin can choose the LOGIN option where they are asked to enter userId&password , and if he/she is a valid user then a admin login page will be displayed.

5.2.2 DATA FLOW DIAGRAM FOR USER LOGIN



After entering to the home page of the website, User can choose the LOGIN option where they are asked to enter user name & password , and if he/she is a valid user then Booking page will be displayed.

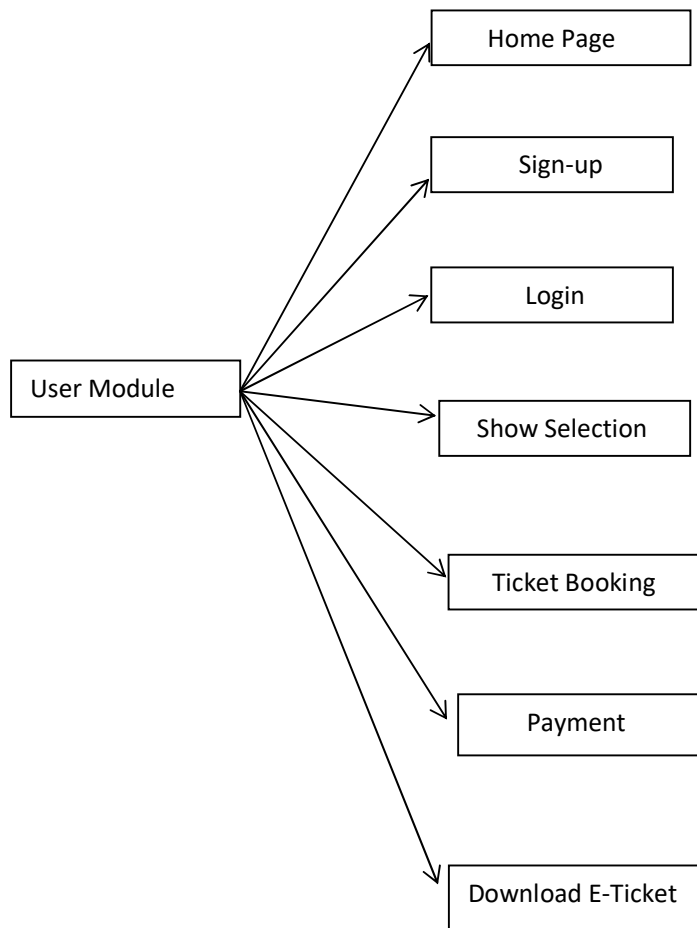
CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 MODULE DESCRIPTION

For Movie Ticket Booking System it is divided into the following Modules:

USER MODULE:



The User module contains Admin home page, signup, login, show selection, Ticket Booking , Payment and Download E-Ticket.

6.2 SOURCE CODE

Backend:

main.py

```
from flask import Flask
from flask import request
from flask import render_template
from models import database
from service import service

app = Flask(__name__)

logged_in_user = ""
logged_in = False

@app.route("/", methods=["GET", "POST"])
def entry():
    global logged_in
    global logged_in_user
    if request.method == "POST":
        # print(request.form)
        if "button" in request.form.keys() and request.form["button"] == "Sign Up":
            return render_template("signup.html")
```



```

elif "button" in request.form.keys() and request.form["button"]
== "Login":

    return render_template("login.html")

elif "button" in request.form.keys() and request.form["button"]
== "Continue as Guest":

    return render_template("form.html", status=logged_in)

elif "Register" in request.form.keys() and
request.form["Register"] == "Register":

    username = request.form.get("username")
    password = request.form.get("password")
    service().createUser(username, password)
    return render_template("login.html")

elif "Login" in request.form.keys() and request.form["Login"] ==
"Login":

    username = request.form.get("username")
    password = request.form.get("password")
    status = service().validateUser(username, password)
    if status:

        logged_in_user = username
        logged_in = True
        return render_template("form.html", status=logged_in)
    else:

        return render_template("login.html", status=logged_in)

```

```

elif "Search" in request.form.keys() and request.form["Search"]
== "Search":

    location = request.form.get("location")

    movie = request.form.get("movie")

    theatre = request.form.get("theatre")

    screen = request.form.get("screen")

    seat = request.form.get("seat")

    res = service().getList(location, movie, theatre, screen, seat)

    return render_template("form.html", output_data=res,
        status=logged_in)

elif "Book" in request.form.keys() and request.form["Book"] ==
"Book":

    location = request.form.get("location")

    movie = request.form.get("movie")

    theatre = request.form.get("theatre")

    screen = request.form.get("screen")

    showtime = request.form.get("showtime")

    seat = request.form.get("seat")

    res = service().bookTicket(location, movie, theatre, screen,
        seat, showtime, logged_in_user)

    if res:

        res = 1

    else:

        res = 2

```

```

        return render_template("form.html", booking_status=res,
                                status=logged_in)

    return render_template("open.html")

if __name__ == "__main__":
    database()
    app.run(host="127.0.0.1", port=8080, debug=True)

```

models.py

```

import sqlite3

class database:
    def __init__(self):
        self.conn = sqlite3.connect('movies.db')
        self.create_database()
        self.populate_database()

    def __del__(self):
        self.conn.commit()
        self.conn.close()

    def create_database(self):
        self.create_table_users()
        self.create_table_movies()

```

```
self.create_table_booking()
```

```
def populate_database(self):
```

```
    self.insert_movies()
```

```
def create_table_users(self):
```

```
    query = """
```

```
        CREATE TABLE IF NOT EXISTS user (  
            id INTEGER PRIMARY KEY AUTOINCREMENT,  
            username TEXT NOT NULL,  
            password TEXT  
        );  
    """
```

```
    self.conn.execute(query)
```

```
def create_table_movies(self):
```

```
    query = """DROP TABLE IF EXISTS Movies"""
```

```
    self.conn.execute(query)
```

```
    query = """
```

```
        CREATE TABLE IF NOT EXISTS Movies (  
            movie_name TEXT ,  
            theatre_name TEXT,  
            location TEXT,
```

```

        screen TEXT,
        showtime TEXT,
        available_seats text,
        id INTEGER PRIMARY KEY AUTOINCREMENT
    );
    """

    self.conn.execute(query)

```

```

def create_table_booking(self):
    query = """
        CREATE TABLE IF NOT EXISTS booking (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_name TEXT,
            movie_name TEXT,
            theatre_name TEXT,
            location_name TEXT,
            showtime TEXT,
            screen TEXT,
            seat_number TEXT
        );
        """

    self.conn.execute(query)

```

```

def insert_movies(self):
    query = """
        INSERT INTO Movies
        (movie_name,theatre_name,location,screen,
        showtime,available_seats)
        VALUES

        ('Viduthalai', 'Surya Cinemas', 'Villupuram', 'A', '10:00',
        '1,2,3,4,5,6,7,8,9,10'),

        ('Vikram', 'Surya Cinemas', 'Villupuram', 'A', '13:00',
        '1,2,3,4,5,6,7,8,9,10'),

        ('PS-2', 'Surya Cinemas', 'Villupuram', 'A', '16:00',
        '1,2,3,4,5,6,7,8,9,10'),

        ('Soorai Potru', 'Surya Cinemas', 'Villupuram', 'A', '19:00',
        '1,2,3,4,5,6,7,8,9,10'),

        ('PS-2', 'Surya Cinemas', 'Villupuram', 'A', '22:00',
        '1,2,3,4,5,6,7,8,9,10'),

        ('Viduthalai', 'Surya Cinemas', 'Villupuram', 'B', '10:00',
        '1,2,3,4,5'),

        ('Vikram', 'Surya Cinemas', 'Villupuram', 'B', '13:00',
        '1,2,3,4,5'),

        ('PS-2', 'Surya Cinemas', 'Villupuram', 'B', '16:00', '1,2,3,4,5'),

        ('Soorai Potru', 'Surya Cinemas', 'Villupuram', 'B', '19:00',
        '1,2,3,4,5'),

        ('PS-2', 'Surya Cinemas', 'Villupuram', 'B', '22:00', '1,2,3,4,5'),

        ('Viduthalai', 'Surya Cinemas', 'Pondicherry', 'A', '11:00',
        '1,2,3,4,5,6,7,8,9,10'),

        ('Vikram', 'Surya Cinemas', 'Pondicherry', 'A', '14:00',
        '1,2,3,4,5,6,7,8,9,10'),
    """

```

```

('PS-2', 'Surya Cinemas', 'Pondicherry', 'A', '17:00',
'1,2,3,4,5,6,7,8,9,10'),

('Soorarai Potru', 'Surya Cinemas', 'Pondicherry', 'A', '20:00',
'1,2,3,4,5,6,7,8,9,10'),

('PS-2', 'Surya Cinemas', 'Pondicherry', 'A', '23:00',
'1,2,3,4,5,6,7,8,9,10');

```

```

"""

```

```

self.conn.execute(query)

```

```

class request:

```

```

    def __init__(self):

```

```

        self.conn = sqlite3.connect("movies.db")

```

```

        self.conn.row_factory = sqlite3.Row

```

```

    def __del__(self):

```

```

        self.conn.commit()

```

```

        self.conn.close()

```

```

    def getMovieByLocation(self, location):

```

```

        query = "select movie_name, theatre_name, location,showtime,
screen, available_seats from Movies where " \

```

```

        f"location = '{location}';"

```

```

result_set = self.conn.execute(query).fetchall()
result = [{column: row[i]
            for i, column in enumerate(result_set[0].keys())}
           for row in result_set]
return result

```

```

def getTheatreByMovies(self, movie_name):
    query = "select movie_name, theatre_name, location, showtime,
                screen, available_seats" \
            " from Movies where " \
            f"movie_name = '{movie_name}';"
    result_set = self.conn.execute(query).fetchall()
    result = [{column: row[i]
                for i, column in enumerate(result_set[0].keys())}
               for row in result_set]
    return result

```

```

def createUser(self, username, password):
    query = f'insert into user ' \
            f'(username, password) ' \
            f'values ("{username}", "{password}")'
    self.conn.execute(query)

```



```

def validateUser(self, username, password):
    query = "select * from user where " \
        f"username = '{username}' and password = '{password}';"
    result_set = self.conn.execute(query).fetchall()
    if len(result_set) == 0:
        return False
    return True

def createEntry(self, location, movie_name, theatre, screen, seat,
showtime, logged_in_user):
    query = "INSERT INTO booking (user_name, movie_name, " \
        "theatre_name, location_name, showtime, screen, " \
        "seat_number)" \
        " VALUES " \
        f"('{logged_in_user}', '{movie_name}', '{theatre}', " \
        f"'{location}', '{showtime}', '{screen}', '{seat}') ;"
    self.conn.execute(query)

def updateMovies(self, location, movie_name, theatre, screen, seat,
showtime):
    query = "update Movies " \
        f"set available_seats = '{seat}' where " \
        f"location = '{location}' " \
        f"and movie_name = '{movie_name}' " \

```

```

f"and theatre_name = '{theatre}' " \
f"and screen = '{screen}' " \
f"and showtime = '{showtime}' ;"

```

```

self.conn.execute(query)

```

```

def bookTicket(self, location, movie_name, theatre, screen, seat,
showtime, logged_in_user):

```

```

    query = "select movie_name, theatre_name, location, showtime,
        screen, available_seats" \
        " from Movies where " \
        f"location = '{location}' " \
        f"and movie_name = '{movie_name}' " \
        f"and theatre_name = '{theatre}' " \
        f"and screen = '{screen}' "\
        f"and showtime = '{showtime}' ;"

```

```

result_set = self.conn.execute(query).fetchall()

```

```

if len(result_set) != 1:

```

```

    return False

```

```

s = result_set[0]["available_seats"]

```

```

li = s.split(",")

```

```

if seat not in li:

```

```

    return False

```

```
self.createEntry(location, movie_name, theatre, screen, seat,
showtime, logged_in_user)

li.remove(seat)

s = ','.join(li)

self.updateMovies(location, movie_name, theatre, screen, s,
showtime)

return True
```

FRONTEND:

signup.html

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Signup Page</title>

</head>

<style>
```

```
body{

    background-
        image:url('https://wallpaperaccess.com/full/3302873.jpg');

    background-repeat: no-repeat;

    background-position:center;
```

```
background-size: 100%;  
margin: 0;  
}  
.logo{  
margin: 0;  
margin-bottom: 50px;  
padding: .5em;  
font-size: 2em;  
min-width: 100px;  
color: white;  
font-family: 'Poppins', sans-serif;  
text-align: center;  
background-color: black;  
}  
.form{  
display: flex;  
flex-direction: column;  
height: 350px;  
width: 350px;  
border: 1px solid rgb(223, 219, 219);  
align-items: center;  
margin: auto;  
margin-top: 100px;
```

```
background-color: rgba(95, 93, 93, 0.773);
border-radius: 25px;
}
input{
width: 200px;
height: 10px;
margin: 10px;
border-radius: 5px;
outline: none;
border: none;
}
.form .btn{
height: 20%;
width: 38%;
margin-bottom: 10%;
border-radius: .5em;
background-color: rgb(93, 192, 238);
color: black;
font-family: 'poppins',sans-serif;
font-weight: 400;
font-size: 1em;
cursor: pointer;
transition: 0.4s ease;
```

```
border: 1px solid black;
}
.username {
    width:55%;
    height: 10%;
    padding: .5em;
    margin-top:10%;
}
.password{
    width:55%;
    height: 10%;
    padding: .5em;
    margin:0;
}
.confirm{
    width:55%;
    height: 10%;
    padding: .5em;
    margin:0;
}
footer{
    height: 30%;
    text-align: center;
```

```

margin-top:90px ;
color:white;
background-color:black;
padding:.25em 1em ;
}
</style>
<body>

    <a><h1 class = "logo">SURYA CINEMAS</h1></a>

<form action="" class="form" method="post">
    <input type="text" class="username" name="username"
        placeholder="UserName" ><br><br>
    <input type="password" class="password" name="password"
        placeholder="Password" ><br><br>
    <input name="confirm password" class="confirm"
        type="password" placeholder="Confirm Password" ><br><br>
    <input type="submit" class="btn" name="Register"
        value="Register">
</form>

<br>

<footer>
    <div class="">&#169; copyright SURYA CINEMAS</div>
</footer>

```

```
</body>
```

```
</html>
```

login.html

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

```
    <title>Login Page</title>
```

```
</head>
```

```
<style>
```

```
body{
```

```
    background-
```

```
    image:url('https://wallpaperaccess.com/full/3302873.jpg');
```

```
    background-repeat: no-repeat;
```

```
    background-position:center;
```

```
    background-size: 100%;
```

```
    margin: 0;
```

```
}
```

```
.logo{
```

```
    margin: 0;
```

```
    margin-bottom: 50px;
```



```
padding: .5em;
font-size:2em;
min-width:100px;
color:white;
font-family: 'Poppins',sans-serif;
text-align: center;
background-color: black;
}

.form{
display: flex;
flex-direction: column;
height: 350px;
width: 350px;
border: 1px solid rgb(223, 219, 219);
align-items: center;
margin: auto;
margin-top: 100px;
background-color: rgba(95, 93, 93, 0.773);
border-radius: 25px;
}
```

```
input{
```

```
width: 200px;
height: 10px;
margin: 10px;
border-radius: 5px;
outline: none;
border: none;
}

.form .btn{
  height: 20%;
  width: 38%;
  margin-bottom: 15%;
  border-radius: .5em;
  background-color: rgb(93, 192, 238);
  color: black;
  font-family: 'poppins',sans-serif;
  font-weight: 400;
  font-size: 1em;
  cursor: pointer;
  transition: 0.4s ease;
  border: 1px solid black;
}

form .btn{
  background-color: transparent;
```

```

}

.username {
    width:50%;
    height: 15%;
    padding: .5em;
    margin-top: 25%;
}

.password{
    width:50%;
    height: 15%;
    padding: .5em;
}

p.ex2{
    color:chartreuse;
    text-align: center;
}

footer{
    height: 30%;
    text-align: center;
    margin-top:60px ;
    color:white;
    background-color:black;
}

```

```

padding:.25em 1em ;
}

</style>
<body>
    <a><h1 class = "logo">SURYA CINEMAS</h1></a>

    <form action="" class="form" method="post">
        <input type="text" class="username" name="username"
            placeholder="UserName"><br><br>
        <input type="password" class="password" name="password"
            placeholder="Enter your password"><br><br>
        <input type="submit" class="btn" name="Login" value="Login">
    </form>

    <br>
    {% if status == False %}
    <p class = "ex2">Invalid username or password</p>
    {% endif %}

    <footer>
    <div class="">&#169 copyright SURYA CINEMAS</div>
    </footer>
</body>
</html>

```

form.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Surya Cinemas</title>
</head>
<style>
body{
  background-
image:url('https://wallpaperaccess.com/full/3302873.jpg');
  background-repeat: no-repeat;
  background-position:center;
  background-size: 100%;
  margin: 0;
}
.logo{
  margin: 0;
  margin-bottom: 50px;
  padding: .5em;
  font-size:2em;
  min-width:100px;
  color:white;
```

```
font-family: 'Poppins',sans-serif;
text-align: center;
background-color: black;
}
.userinfo{
    align-items: center;
    justify-content: center;
    margin:auto;
    border: 1px solid white;
    padding:10px;
}
p{
    color:white;
}
label{
    color: white;
    display: inline-block;
    float: left;
    clear: left;
    width: 80px;
    text-align:right;
}
input{
```

```
    display: inline-block;
    float:left;
}
.container{
    text-align: center;
}
form{
    width: 400px;
    padding: .5em;
    align-items: center;
    margin: 0;
    margin-top: 20px;
    background-color: rgba(95, 93, 93, 0.773);
    border-radius: 10px;
}
table{
    color:white;
    align-items: center;
    justify-content: center;
    margin:auto;
}
.click{
    color: rgb(235, 218, 66);
```

```

}
div .btn{
    display: flex;
    flex-wrap: wrap;
    flex-direction: row;
    row-gap: 10px;
    height: 10%;
    width: 18%;
    gap: 10px;
    margin-top: 10%;
    border-radius: .5em;
    background-color: white;
    color: black;
    border: 1px solid black;
}
footer{
    height: 30%;
    text-align: center;
    margin-top: 118px ;
    color: white;
    background-color: black;
    padding: .25em 1em ;
}

```



```

</style>

<body>

<a><h1 class = "logo">SURYA CINEMAS</h1></a>

    <center><p><b>You can search by Location or by Movie</b><br>

    Once you have found the perfect watch <b>input all the values and
    select the Book button</b> <br>

    Thanks for using our product...</p><br><br></center>

<form action="" class="userinfo" method="post">

    <label>Location : &nbsp;</label><input type="text"
name="location" ><br><br>

    <label>Movie : &nbsp;</label> <input type="text"
name="movie"><br><br>

    <label>Theatre: &nbsp;</label><input type="text"
name="theatre"><br><br>

    <label>Screen :&nbsp;</label><input type="text"
name="screen"><br><br>

    <label>Showtime : &nbsp;</label><input type="text"
name="showtime"><br><br>

    <label>Seat : &nbsp;</label><input type="text"
name="seat"><br><br>

    <div class = "container">

        <input type="submit" class="btn" name="Search"
value="Search"></div>

        {% if status %}

        <div class = "container">

            <input type="submit" class="btn" name="Book" value="Book">

```

```

</div>

{% endif %}

</form>

<br><br><br>

{% if output_data %}

<table>

  <thead>

    <tr>

      <th>Movie</th>

      <th>Theatre</th>

      <th>Location</th>

      <th>Showtime</th>

      <th>Screen</th>

      <th>Available Seats</th>

    </tr>

  </thead>

  <tbody>

    {% for row in output_data %}

      <tr>

        <td>{{row["movie_name"]}}</td>

        <td>{{row["theatre_name"]}}</td>

        <td>{{row["location"]}}</td>

        <td>{{row["showtime"]}}</td>


```

```

        <td>{{row["screen"]}}</td>
        <td>{{row["available_seats"]}}</td>
    </tr>
    {% endfor %}
</tbody>
</table>
{% endif %}<br><br><br><br>
<center>
{% if booking_status == 1 %}
    <p><b>Seat is Available...</b></p>
    <p>Now, You have to Pay Rs.100 to Book Your Ticket...</p>
    <p>To Pay <a href="{{url_for('static',filename='ticket.html')}}"
    class="click">Click here</a></p>
{% elif booking_status == 2 %}
    <p>Booking is Unsuccessful!!!</p>
{% endif %}
</center>
<footer>
    <div class="">&#169 copyright SURYA CINEMAS</div>
</footer>
</body>
</html>

```

CHAPTER 7

SYSTEM TESTING

The aim of the system testing process was to determine all defects in our project .The program was subjected to a set of test inputs and various observations were made and based on these observations it will be decided whether the program behaves as expected or not.

Our Project went through two levels of testing

1.Unit testing

2.Integration testing

7.1 UNIT TESTING

Unit testing is undertaken when a module has been created and successfully reviewed .In order to test a single module we need to provide a complete environment i.e. besides the module we would require

- The procedures belonging to other modules that the module under test calls
- Non local data structures that module accesses
- A procedure to call the functions of the module under test with appropriate Parameters

Unit testing was done on each and every module that is described under module description of chapter 6

7.2 INTEGRATION TESTING

In this type of testing we test various integration of the project module by providing the input. The primary objective is to test the module interfaces in order to ensure that no errors are occurring when one module invokes the other module.

CHAPTER 8

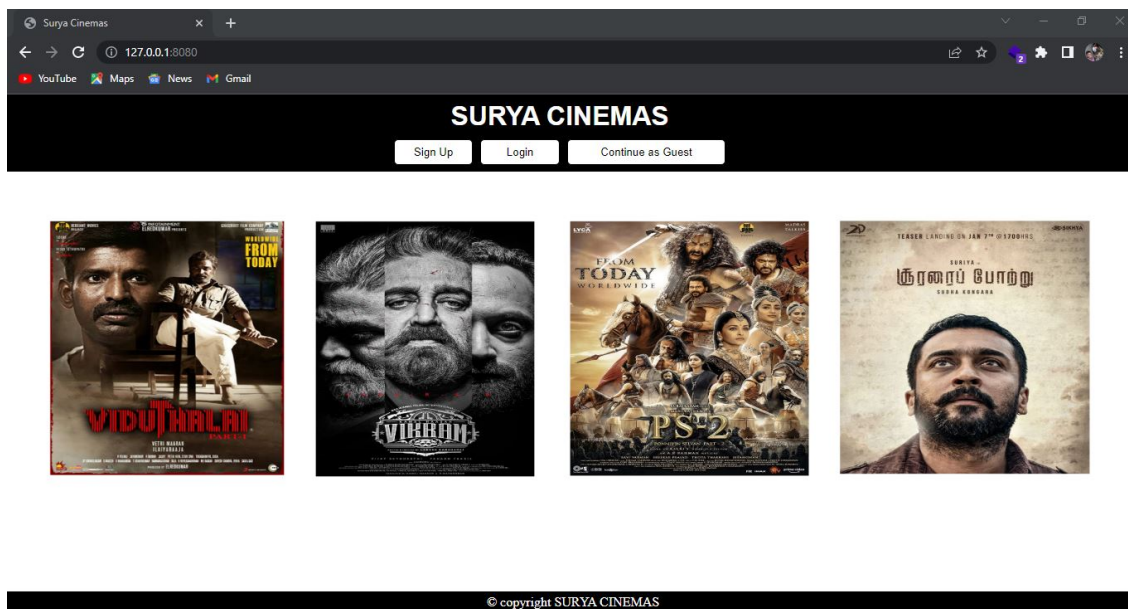
SCREENSHOTS

Run main.py

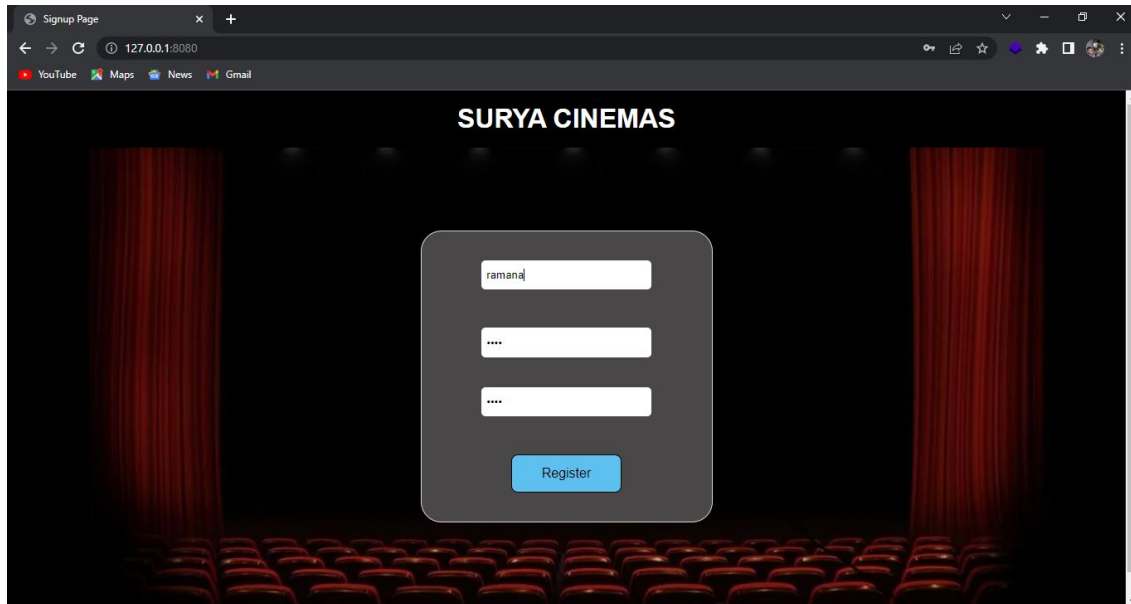
```
C:\Windows\System32\cmd.exe - python main.py
Microsoft Windows [Version 10.0.19045.2913]
(c) Microsoft Corporation. All rights reserved.

E:\movieticket>python main.py
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:8080
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 141-724-250
```

Home page:

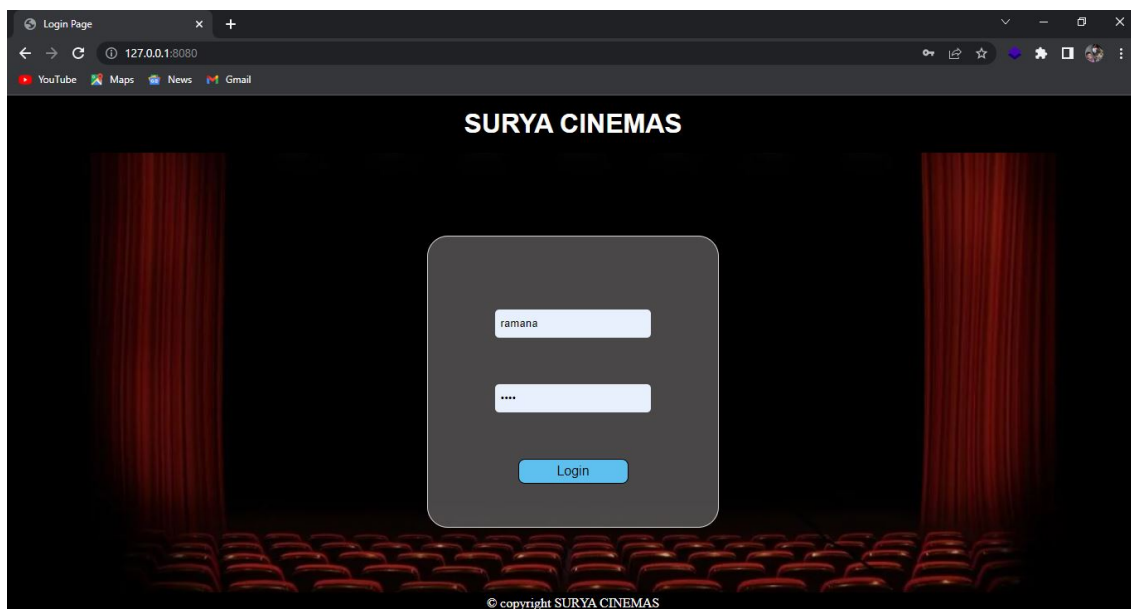


Signup Page:



A screenshot of a web browser displaying the 'Signup Page' for 'SURYA CINEMAS'. The browser's address bar shows '127.0.0.1:8080'. The page features a dark background with a red curtain and rows of red seats. In the center, there is a light gray rounded rectangle containing three input fields: the first is labeled 'ramana', the second and third are masked with four dots. Below these fields is a blue 'Register' button.

Login Page:



A screenshot of a web browser displaying the 'Login Page' for 'SURYA CINEMAS'. The browser's address bar shows '127.0.0.1:8080'. The page features a dark background with a red curtain and rows of red seats. In the center, there is a light gray rounded rectangle containing two input fields: the first is labeled 'ramana', the second is masked with four dots. Below these fields is a blue 'Login' button. At the bottom center of the page, there is a small copyright notice: '© copyright SURYA CINEMAS'.

Show Selection:

SURYA CINEMAS

You can search by Location or by Movie
Once you have found the perfect watch input all the values and select the Book button
Thanks for using our product..

Location :

Movie :

Theatre :

Screen :

Showtime :

Seat :

Movie	Theatre	Location	Showtime	Screen	Available Seats
Viduthalai	Surya Cinemas	Pondicherry	11:00	A	1,2,3,4,5,6,7,8,9,10
Vikram	Surya Cinemas	Pondicherry	14:00	A	1,2,3,4,5,6,7,8,9,10
PS-2	Surya Cinemas	Pondicherry	17:00	A	1,2,3,4,5,6,7,8,9,10
Soorai Potru	Surya Cinemas	Pondicherry	20:00	A	1,2,3,4,5,6,7,8,9,10

Ticket Booking:

SURYA CINEMAS

You can search by Location or by Movie
Once you have found the perfect watch input all the values and select the Book button
Thanks for using our product..

Location :

Movie :

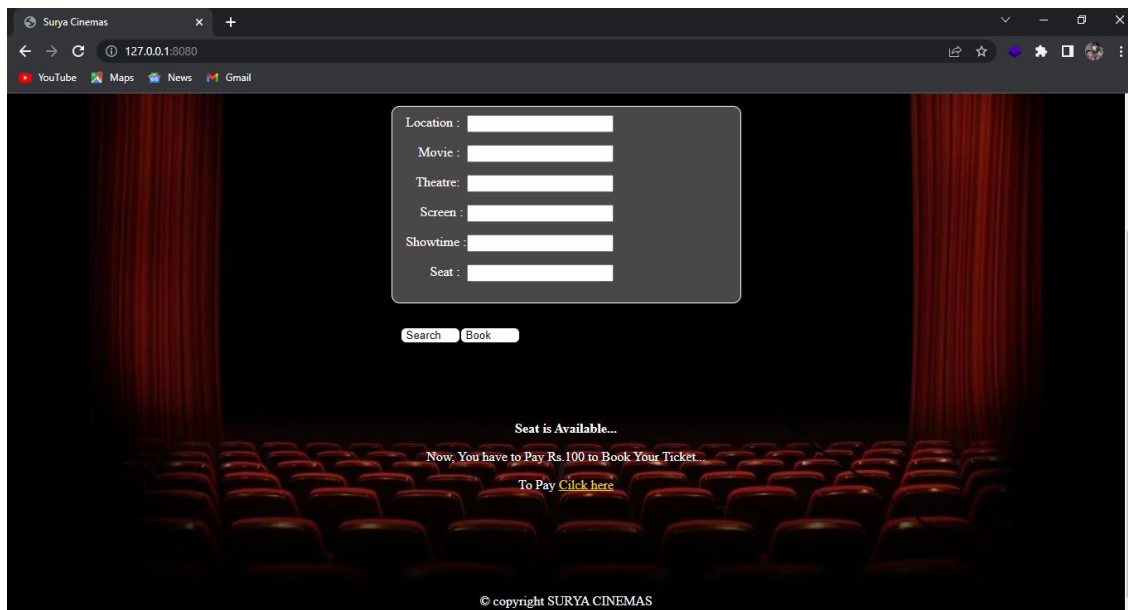
Theatre :

Screen :

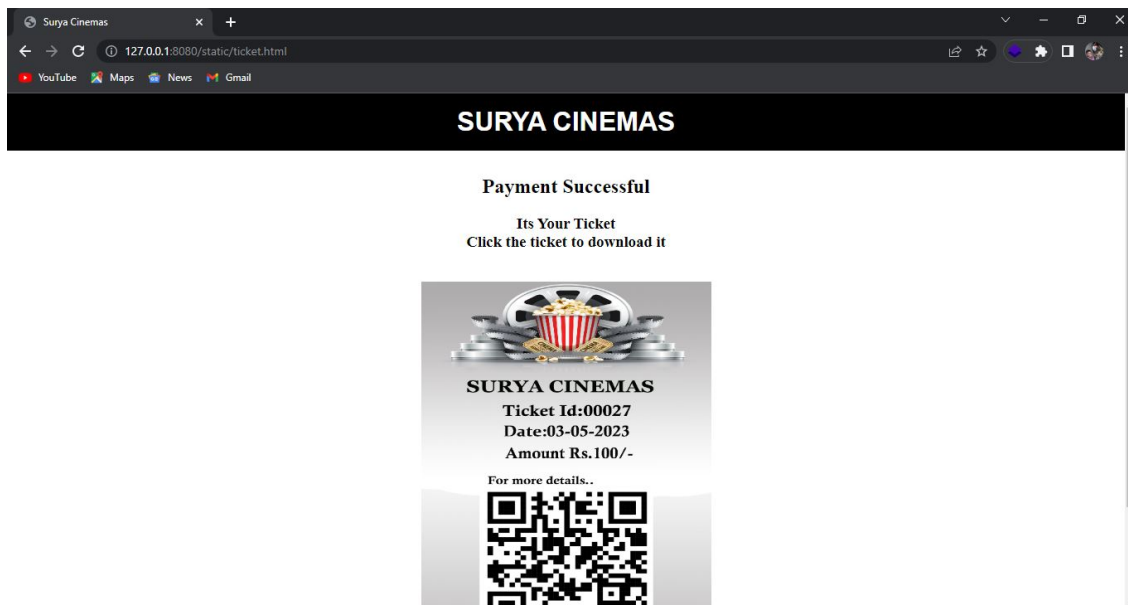
Showtime :

Seat :

Movie	Theatre	Location	Showtime	Screen	Available Seats
Viduthalai	Surya Cinemas	Pondicherry	11:00	A	1,2,3,4,5,6,7,8,9,10
Vikram	Surya Cinemas	Pondicherry	14:00	A	1,2,3,4,5,6,7,8,9,10
PS-2	Surya Cinemas	Pondicherry	17:00	A	1,2,3,4,5,6,7,8,9,10
Soorai Potru	Surya Cinemas	Pondicherry	20:00	A	1,2,3,4,5,6,7,8,9,10



Download Ticket:



CHAPTER 9

CONCLUSION & FUTURE SCOPE

9.1 CONCLUSION

In conclusion, the movie ticket booking system project is a valuable addition to the cinema industry, providing customers with a convenient and efficient way to book movie tickets. The system has many advantages, such as reducing waiting times, providing real-time seat availability information, and enabling customers to book tickets from the comfort of their homes. It also streamlines the ticketing process for cinema halls, resulting in improved operational efficiency and reduced costs.

The project has been successful in meeting its objectives and has the potential for future growth and development. Integration of advanced features like personalized recommendations, virtual seating arrangements, and payment options can enhance the user experience further. Additionally, the system can be expanded to include additional services such as food and beverage ordering, loyalty programs, and special event ticketing.

Overall, the movie ticket booking system project has the potential to revolutionize the movie-going experience for customers and cinema halls alike, making it a valuable contribution to the cinema industry.

9.2 FUTURE SCOPE

- Integration with social media platforms - The system can be integrated with social media platforms to allow users to share their booking details, movie reviews, and recommendations with friends and family. This can increase the user engagement and promote the movie-going experience.
- Personalized recommendations - The system can be enhanced to provide personalized movie recommendations based on the user's past booking history and preferences. This can improve the user experience and increase customer satisfaction.
- Virtual seating arrangements - The system can be developed to include virtual seating arrangements, allowing users to choose their preferred seats before booking. This can enhance the user experience and help cinema halls optimize their seating arrangements.
- Payment options - The system can be expanded to include more payment options, such as mobile wallets, UPI, and cryptocurrencies, to provide users with more flexibility and convenience.

CHAPTER 10

REFERENCES

1. Programming Python: Powerful Object-Oriented Programming (4th Edition) by Mark Lutz - available at: <http://bilal-qudah.com/mm/Programming%20Python%20Fourth%20Edition.pdf>
2. Flask Web Development by Miguel Ginberg - available at: https://coddyschool.com/upload/Flask_Web_Development_Developing.pdf
3. HTML & CSS: Design and Build Web Sites Book by Jon Duckett - available at: <https://wtf.tw/ref/duckett.pdf>
4. <https://docs.python.org/3/tutorial/>
5. <https://www.w3schools.com/python/>
6. <https://www.w3schools.com/html/>
7. <https://www.w3schools.com/w3css/defaultT.asp>
8. <https://www.w3schools.com/sql/>
9. <https://www.tutorialspoint.com/flask/index.htm>
10. Web development and application development by Ivan Byross BPB publications